**VISOKO UČILIŠTE ALGEBRA**

ZAVRŠNI RAD

**IMPLEMENTACIJA RJEŠENJA ZA VOĐENJE I IZVOĐENJE TRENINGA NA UDALJENOSTI**

Filip Besednik

Zagreb, veljača 2021.

*„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.*

*U Zagrebu, datum.*

***Ime i Prezime***

**Predgovor**

Zahvaljujem se svim profesorima i asistentima koji su prenosili svoje znanje na mene i ostale kolege s Visokog učilišta Algebra. Zahvaljujem se svojim kolegama s Visokog učilišta Algebra na međusobnoj potpori i pomoći u usvajanju i razmjenjivanju različitih znanja. Zahvaljujem se svom mentoru Danielu Beleu na pomoći i savjetima tijekom izrade završnog rada. Zahvaljujem se svojim roditeljima Gordani i Damiru, sestri Petri i zaručnici Dori na potpori tijekom studija.

**Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi**

**Sažetak**

S obzirom na novonastalu situaciju gdje je socijalno distanciranje izuzetno važno, ljudi se manje bave tjelesnim aktivnostima, teretane su zatvorene, a samim time treneri ostaju bez posla. Cilj ovog završnog rada je omogućiti jednostavnu komunikaciju korisnika s trenerom u svrhu povećanja tjelesne aktivnosti iz udobnosti doma. Iako je aplikacija namijenjena specifično za vrijeme socijalnog distanciranja, od velike koristi može biti i kada prestanu spomenute mjere. Treneri mogu zaraditi dodatan prihod i imati više klijenata odjednom s obzirom na to da nije potrebna fizička prisutnost tijekom treninga. S druge strane, korisnici mogu dobiti savjete i pomoć privatnih trenera uz manje troškove. Rezultat je jednostavno slaganje vježbi, treninga i rasporeda od strane trenera, te pregled istih od strane korisnika.

**Ključne riječi:** android, aplikacija, socijalno distanciranje, trening, trener

**Abstract**

Given the new situation where social distancing is extremely important, people are engaging less in physical activities, gyms are closed, and thus coaches lose their jobs. The aim of this final work is to enable easy communication of the user with the trainer to increase physical activity from the comfort of home. Although the application is intended specifically during social distancing, it can also be of great benefit when these measures cease. Coaches can earn extra income and have multiple clients at once since no physical presence is required during training. On the other hand, users can get advice and help from private trainers at a lower cost. The result is a simple arrangement of exercises, training and schedules by the trainer, and a review of the same by the user.

**Keywords:** android, application, social distancing, training, coach

**Sadržaj**

[1. Uvod 3](#_Toc61859337)

[2. Potreba za aplikacijom 4](#_Toc61859338)

[2.1. Problematika izvođenja vježbi u vrijeme socijalnog distanciranja 4](#_Toc61859339)

[2.2. Opis profila ispitanika 4](#_Toc61859340)

[2.3. Provođenje ankete 5](#_Toc61859341)

[2.4. Prijedlog rješenja na temelju ankete 8](#_Toc61859342)

[2.5. Posebnosti ovog rješenja u odnosu na postojeće 8](#_Toc61859343)

[3. Korištene tehnologije 9](#_Toc61859344)

[3.1. Android platforma 9](#_Toc61859345)

[3.1.1. Android ekosustav 10](#_Toc61859346)

[3.1.2. Prezentacijski sloj 11](#_Toc61859347)

[3.1.3. Životni ciklus Aktivnosti 11](#_Toc61859348)

[3.2. *Entity* radni okvir 12](#_Toc61859349)

[3.3. Microsoft SQL Server 13](#_Toc61859350)

[4. Arhitektura sustava 15](#_Toc61859351)

[4.1. Poslužiteljski sloj 16](#_Toc61859352)

[4.1.1. Baza podataka 17](#_Toc61859353)

[4.1.2. Web servis 20](#_Toc61859354)

[4.2. Klijentski sloj 22](#_Toc61859355)

[4.2.1. Komunikacija aplikacije s web servisom 23](#_Toc61859356)

[5. Implementacija programskog rješenja 24](#_Toc61859357)

[5.1. Servisni dio sustava 24](#_Toc61859358)

[5.1.1. Baza podataka 24](#_Toc61859359)

[5.1.2. Web servis 27](#_Toc61859360)

[5.2. Klijentski dio sustava 33](#_Toc61859361)

[5.2.1. Funkcionalnosti mobilne aplikacije 33](#_Toc61859362)

[5.2.2. Opis korisnika 43](#_Toc61859363)

[6. Analiza uspješnosti aplikacija nakon produkcije 45](#_Toc61859364)

[Zaključak 46](#_Toc61859365)

[Popis kratica 47](#_Toc61859366)

[Popis slika 48](#_Toc61859367)

[Popis tablica 50](#_Toc61859368)

[Popis kôdova 51](#_Toc61859369)

[Literatura 52](#_Toc61859370)

# Uvod

Sustav *RemoteTrainer* nastoji riješiti problem koji je uzrokovan mjerama socijalnog distanciranja u vrijeme pandemije. Timske tjelesne aktivnosti su zabranjene, rekreativne aktivnosti u sportovima su obustavljene do danjeg, teretane su zatvorene. Jedino profesionalni sportaši i dalje nastavljaju trenirati, ali individualno.

Prvi od ciljeva sustava je potaknuti, promovirati i omogućiti odrađivanje treninga od doma uz pomoć trenera. Sljedeći cilj je omogućiti trenerima stabilan prihod u vrijeme kada ga klasičnim obavljanjem svoje struke ne mogu ostvariti. Napokon, u konačnici, sustav može poslužiti i samom promoviranju treniranja i zdravijeg života.

Sustav troslojne arhitekture sastoji se od mobilne aplikacije za Android uređaje, baze podataka i aplikacijskog programskog sučelja kao most za komuniciranje između aplikacije i baze podataka. Za svaki od dijelova sustava detaljnije je opisana njihova arhitektura, implementacija i komunikacija između njih, u poglavljima koje slijede.

Ideje temeljnih funkcionalnosti ovog sustava su osmišljene i potom realizirane nakon osvrta na riješene ankete od strane trenera i korisnika, uzimajući u obzir njihove savjete o tome što im je najpotrebnije. Glavne funkcionalnosti sustava opisane su u poglavlju **Funkcionalnosti mobilne aplikacije**.

Znanje i podaci potrebni za izradu ovog sustava dobiveni su tijekom školovanja na Visokom učilištu Algebra, te dodatno iz dokumentacija korištenih tehnologija i knjiga navedenih u literaturi.

# Potreba za aplikacijom

Zbog pandemije uzrokovane SARS-CoV-2 virusom došlo je do potrebe za brojnim novim sustavima. *RemoteTrainer* sustav dolazi kao rješenje problema smanjenja fizičke aktivnosti tijekom socijalnog distanciranja. Odavno je poznato kako je fizička aktivnost od izuzetne važnosti, kako za fizičko, tako i za psihičko zdravlje. Sustav opisan u ovom radu pruža mogućnost personaliziranih treninga sastavljenih od profesionalnog trenera, bez potrebe za izravnim kontaktom između trenera i korisnika. Potreba za ovakvim sustavom dodatno je potvrđena u anketi provedenoj nad trenerima i osobama koje su se bavile tjelesnim aktivnostima, kao i osobama koje su se planirale početi baviti tjelesnim aktivnostima neposredno prije pandemije.

## Problematika izvođenja vježbi u vrijeme socijalnog distanciranja

Manjak motivacije i nedovoljno znanja o kineziološkoj struci su se pokazali kao najčešći uzroci smanjenja ili potpunog prestanka izvođenja tjelesnih aktivnosti. Potrebno je napomenuti kako je kontakt s trenerom za pravilno i kvalitetno izvođenje treninga od suštinske važnosti, što je otežano, ako ne i nemoguće tijekom novonastalih okolnosti. Bez stručne pomoći ili motivacije od strane obrazovanog trenera, rekreativci odustaju od treniranja ili se uopće ne počinju baviti tjelesnim aktivnostima. Nadalje, veliki problem predstavlja i nedostatak kvalitetne opreme pojedinca, pa su od izuzetne važnosti savjeti i sugestije od stane stručne osobe, kako bi se omogućilo kvalitetno obavljanje treninga u okolini koja nije u potpunosti opremljena.

## Opis profila ispitanika

Ispitanici se mogu podijeliti u dvije skupine: treneri i potencijalni korisnici usluga trenera. Ispitani treneri su iz sportova kao što su: taekwondo, judo, skijanje te obrazovane osobe kineziološke struke. Potencijalni korisnici usluga trenera su osobe koje su se nakon mjera socijalnog distanciranja prekinule baviti tjelesnim aktivnostima i osobe koje se planiraju početi baviti tjelesnim aktivnostima.

## Provođenje ankete

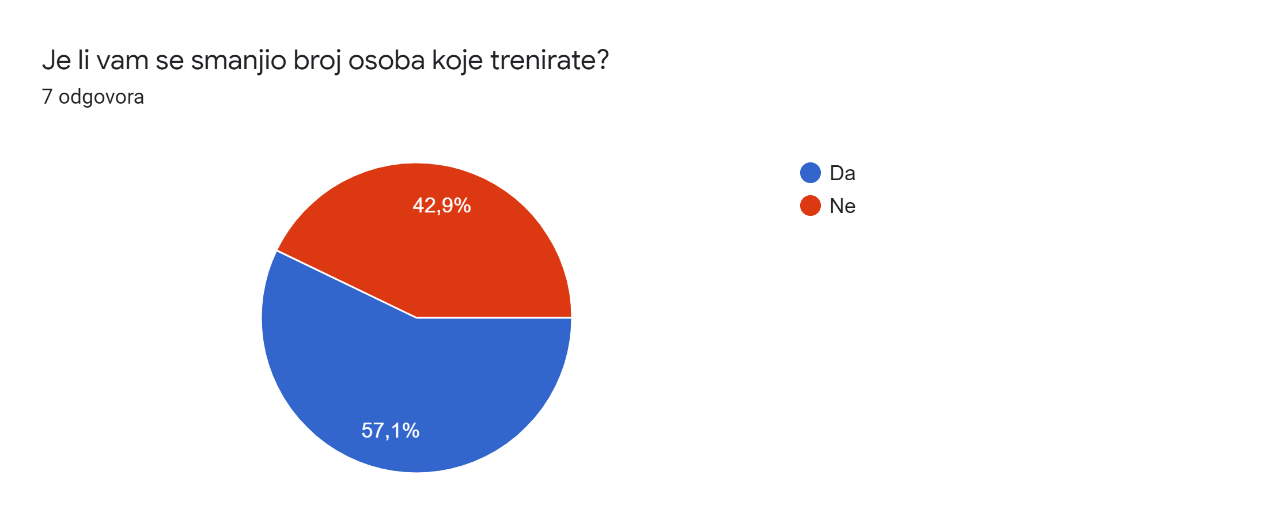
Anketa je napravljena uz pomoć Google obrazaca i proslijeđena trenerima i potencijalnim korisnicima usluga trenera. Anketa je provedena u prosincu 2020. godine tijekom drugog vala pandemije SARS-CoV-2 virusa i novih mjera izrečenih od strane stručnog stožera za civilnu zaštitu. Svi ispitanici su ispunili anketu i dali važan uvid u problem i potencijalna rješenja za izvođenje vježbi tijekom socijalnog distanciranja. Anketa je pomogla u izoštravanju funkcionalnosti koje su implementirane u *RemoteTrainer* sustav, o kojima će biti riječ u narednom poglavlju. U anketi je sudjelovalo ukupno 18 ispitanika. Riječ je o sedam trenera i 11 korisnika. Slijedi grafički prikaz anketnih rezultata trenera i tekstualni prikaz anketnih rezultata korisnika.



Slika . Prvo pitanje za trenere iz ankete



Slika . Drugo pitanje za trenere iz ankete



Slika . Treće pitanje za trenere iz ankete



Slika . Četvrto pitanje za trenere iz ankete



Slika . Peto pitanje za trenere iz ankete

Slike prikazane iznad pokazuju postavljena pitanja trenerima u anketi i njihove rezultate. Većini trenera se smanjio broj treninga koje vode i broj osoba kojima vode treninge. Svi treneri su zainteresirani za mogućnost kakvu *RemoteTrainer* sustav pruža. Zadnje postavljeno pitanje u anketi koje glasi „**Imate li kakvih savjeta što bi aplikacija trebala imati**“ dalo je par odličnih ideja i savjeta iz kojih su već smišljene funkcionalnosti dodatno zaokružene. Neke od ideja su:

* „bilo bi dobro da se mogu izraditi vlastite vježbe koje po potrebama korisnika biramo koje želimo slati“
* „izrada svojih vježbi, treninga koji se mogu ponoviti par puta tjedno ili koristiti za sljedeći tjedan i sastavljanje nekakvog rasporeda s tim treninzima“
* „*uplodanje* videa kako se neka vježba izvodi“

Korisnici usluga trenera su u anketi imali postavljeni sljedeća pitanja:

* Jeste li trenirali prije pandemije ili ste planirali početi neposredno prije?
* Jesu li nove mjere prekinule ili smanjile vaše tjelesne aktivnosti?
* Biste li htjeli savjete stručnih trenera?
* Jeste li zainteresirani dostupnost stručnog trenera za sastavljanje treninga i tjednog rasporeda isključivo putem aplikacije?
* Imate li kakvih savjeta što bi aplikacija trebala imati?

Iz rezultata je zanimljivo izdvojiti kako su svi korisnici prekinuli ili smanjili svoje tjelesne aktivnosti u odnosu na vrijeme prije socijalnost distanciranja, što dodatno naglašava važnost ovakve aplikacije. Kod korisnika su savjeti uglavnom slični onima od strane trenera, jedino s korisničke strane. Međutim, jedan savjet se pojavio više puta, a to je *uplodanje* videa koji prikazuje pravilno izvođenje vježbi korak po korak.

## Prijedlog rješenja na temelju ankete

Kao jedno od rješenja za probleme opisane u poglavlju **Problematika izvođenja vježbi u vrijeme socijalnog distanciranja** i na temelju savjeta od strane anketnih ispitanika, izrađena je i opisana u ovom radu aplikacija *RemoteTrainer.* Aplikacijase, u grubo, može podijeliti na sustav za korisnike i sustav za trenere. Okvirno, sustav za korisnike omogućava pronalazak trenera, zahtijevanje poslovnog odnosa, pregled rasporeda, pregled treninga za pojedini dan u rasporedu, pregled vježbi za pojedini trening. S druge strane, sustav za trenere omogućava pregled svih aktivnih zahtjeva za poslovnu suradnju, pregled svojih korisnika, sastavljanje vježbi, sastavljanje treninga i sastavljanje rasporeda za pojedinog korisnika. Međutim, bitno je napomenuti da je aplikacija razvijena na modularan način koji omogućuje laganu nadogradnju, već prema potrebama koje će se pojaviti samim korištenjem aplikacije.

## Posebnosti ovog rješenja u odnosu na postojeće

Na tržištu već postoje aplikacije namijenjene za privatne trenere i osobe koje treniraju. Neke od najpoznatijih uključuju: *Practice Better*, *True Coach*, *The Training Notebook*. Postojeće aplikacije se uglavnom mogu podijeliti u dvije skupine

* aplikacija za privatne trenere u svrhu praćenja i rukovođenja svojih klijenata na jednom mjestu
* aplikacija za praćenje i rukovođenje vlastitih treninga za osobe koje se bave tjelesnim aktivnostima.

Sustav *RemoteTrainer* spaja te dvije skupine i omogućuje komunikaciju trenera i korisnika preko mobilnog uređaja, rukovođenje i pregled rasporeda, treninga i vježbi. Drugim riječima, između trenera i osoba koje se bave tjelesnim aktivnostima nije više potreban izravan fizički kontakt, nego se sva interakcija odvija preko sustava opisanog u ovom radu.

# Korištene tehnologije

Prilikom implementacije slojeva sustava korištene su različite tehnologije. Prezentacijski sloj je izrađen na Android platformi i programiran u Java programskom jeziku, u razvojnoj okolini Android Studio. Sloj poslovne logike je izrađen korištenjem ASP.NET Web API radnog okvira, u *Visual Studio* integriranom razvojnom okruženju te korištenjem C# programskog jezika. Podatkovni sloj je izrađen uz pomoć Microsoft SQL Servera i T-SQL upitnog jezika, korištenjem *Entity* radnog okvira.

Za prijenos podataka između prezentacijskog i poslovnog sloja korišten je JSON format. Napokon, za realizaciju plaćanja unutar aplikacije korišten je PayPal servis. PayPal je mrežni financijski servis koji omogućuje brze i sigurne transakcije s jednog računa na drugi. Servis sačinjavaju javne API točke koje programer može koristiti u svojoj aplikaciji kako bi plaćanje bilo integrirano unutar same aplikacije, bez potrebe za odlaskom na samu PayPal aplikaciju.

## Android platforma

Android platforma podržava izradu aplikacija za uređaje koji koriste modificirani *Linux kernel.* Android platforma je prvi put predstavljena 2007. godine od strane *Open Handset Alliance* kompanije. *Nativne* aplikacije na Android platformi programiraju se u Java ili Kotlin programskim jezicima. Međutim, važno je napomenuti kako se ne koristi Java virtualna mašina tijekom izrade (engl. *build*) aplikacije u .apk datoteku, već se Java klase pretvaraju u Dalvik izvršne datoteke i pokreću se na Dalvik virtualnoj mašini. [1]

Osim Dalvik virtualne mašine aplikacije mogu biti pokrenute s *Android Runtime* izvedbom. ART je nasljednik Dalvik virtualne mašine i kompatibilan je sa Dalvikom, što znači da aplikacije izrađene za Dalvik rade i prilikom pokretanja s ART. ART koristi Dalvik pokretački format i d*ex bytecode*.

Android platforma je platforma otvorenog kôda. Također, bitno je naglasiti kako programeri mogu doprinijeti svim objavljenim verzijama Android platforme, no ne i verzijama koje su trenutno u izradi.

### Android ekosustav

Android ekosustav omogućuje programerima na Android platformi jednostavnu i efikasnu izradu aplikacija za Android uređaje. Android ekosustav sastoji se od alata navedenih u nastavku.

Komponente korisničkog sučelja su gotove komponente, spremne za korištenje i implementaciju od strane programera te pružaju usluge kao što su: ispis teksta, unos teksta, padajući izbornik, komponenta za sliku, gumb, gumb sa slikom, različite liste entiteta i slično. Neke od spomenutih komponenta su: *TextView*, *EditView, Spinner, RecyclerView, ListView, ImageView, Button, ImageButton.*[2]

Jedna od najvažnijih korištenih komponenta u *RemoteTrainer* aplikaciji je *RecyclerView*. On omogućuje jednostavno prikazivanje velikih skupina podataka. Programer definira kako će se svaki podatak u listi prikazati. *RecyclerView* ne uništava *view* komponente prilikom pomicanja liste već ih reciklira i ponovno iskorištava za prikaz novih podataka u listi. Prilikom korištenja *RecyclerView* komponente potrebno je implementirati dvije klase. *ViewHolder* klasa sadrži *layout* datoteku koja definira prikaz svakog od predmeta iz liste. *Adapter* klasa kreira *ViewHolder* klase po potrebi i postavlja im podatke. [2]

Namjera (engl. *Intent*) predstavlja komponentu Android operacijskog sustava koja omogućuje pokretanje aktivnosti (engl. *Activity*) i ostalih komponenata unutar aplikacije, ali i drugih aplikacija na Android uređaju koje to dopuštaju. Naime, namjere omogućavajulako korištenje funkcionalnosti drugih aplikacija, kao što je aplikacija kamere za uzimanje slika i videa. [2] Spomenuto svjedoči o orijentaciji same platforme na slobodu povezivanja komponenata različitih aplikacija u složeni ekosustav.

Servisi (engl. *Service*) su komponente namijenjene izvršavanju pozadinskih zadataka i ne zahtijevaju korisničko sučelje. Nakon što je servis pokrenut, on može u pozadini nastaviti s radom i nakon što je korisnik ugasio aplikaciju. Servisi, primjerice, mogu slati obavijesti korisniku, obavljati mrežne transakcije, komunicirati s davateljem sadržaja (engl. *Content Provider*) i slično. [2]

Emitirani prijamnik (engl. *Broadcast Receiver*) je komponenta koja se koristi se za slanje i primanje poruka od ostalih Android aplikacija te samog operacijskog sustava. Na primjer, određenu aplikaciju može zanimati kada se uređaj pokrene, te se ona pretplati (engl. *subscribe*)na taj događaj (engl. *event*). Nadalje, mogu se izraditi prijemnici prilagođeni posebnim događajima, kao što je događaj primitka datoteke ili vlastita SMS aplikacija. [2]

Davatelj sadržaja je dio ekosustava koji upravlja pristupom zaštićenim podacima. Glavna mu je svrha omogućiti pristupanje od strane ostalih aplikacija, i to putem *provider client* objekata. Time se klijentu osigurava sučelje prema podacima aplikacije na univerzalan način, poštujući sve sigurnosne aspekte. [2]

Aktivnost predstavlja jednu od glavnih komponenta Android ekosustava i sastoji se od prezentacijskog i poslovnog sloja. Pojednostavljeno, aktivnost predstavlja jedan ekran koji je definiran korisničkim sučeljem od strane programera. Aktivnost karakterizira vlastiti životni ciklus koji je realiziran metodama, pozvanim od strane operativnog sustava, prilikom promjena stanja kao što su: pokretanje i gašenje aktivnosti. Životni ciklus aktivnosti objašnjen je detaljno u nastavku poglavlja.

### Prezentacijski sloj

Prezentacijski sloj definiran je aktivnostima. Svaka aktivnost ima svoj raspored (engl. *layout*)napisan u *xml* datoteci. Unutar *layout* datoteke najčešće se nalazi krovni element koji predstavlja tip rasporeda (engl. *layout*). Raspored, kao objekt, je hijerarhijska struktura komponenti korisničkog sučelja i definiran je od strane programera. Neki od tipova rasporeda su sljedeći:

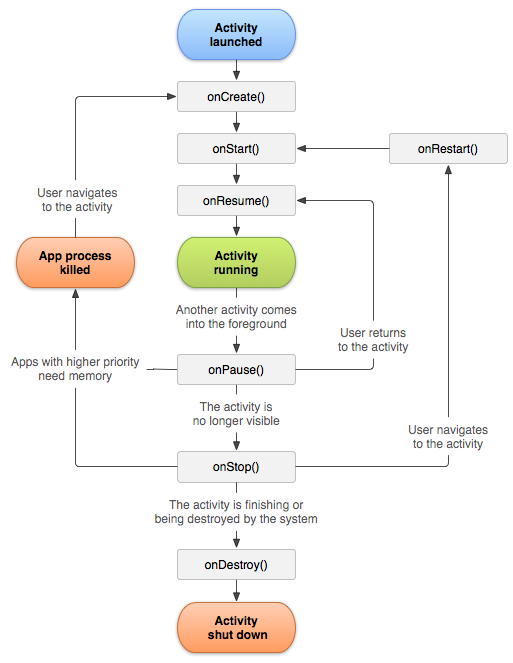
* linearni (engl. *linear*)*,* raspored koji slaže svoju djecu u jedan red, horizontalni ili vertikalni
* relativni (engl. *relative*) raspored koji omogućuje programeru da poslaže komponente relativne jedne drugima
* tablični (engl. *table*)raspored slaže komponente u redove i kolone
* apsolutni (eng. *absolute*) raspored omogućuje specificiranje točnu lokaciju svake komponente na ekranu
* okvirni (engl. *frame*) raspored rezervira mjesto za prikazivanje jednog pogleda (engl. *view*)

Napokon, raspored ograničenja (engl. *constraint layout*) sličan je relativnom rasporedu jer slaže komponente relativne jedni drugima, ali je fleksibilniji i lakši za korištenje iz uređivača izgleda (engl. *layout editor*) Android Studio razvojne okoline.

### Životni ciklus Aktivnosti

Važna stavka poslovnog sloja aktivnosti je njegov životni ciklus. Životni ciklus definiran je pozivima metoda od strane operativnog sustava, prilikom prijelaza aktivnosti kroz različita stanja:

* kreiranje – pozvana je *onCreate()* metoda prilikom stvaranja aktivnosti [3]
* početak – pozvana je *onStart()* metoda kada je aktivnost spreman za prikaz korisniku [3]
* nastavak – pozvana je *onResume()* metoda kada je aktivnost prikazana [3]
* pauziranje – pozvana je *onPause()* metoda kada aktivnost izgubi fokus [3]
* prekid – pozvana je *onStop()* metoda aktivnost više nije prikazana [3]
* ponovno pokretanje – pozvana je *onResume()* metoda kada aktivnost bude ponovno prikazana [3]
* gašenje – pozvana je *onDestroy()* metoda prilikom gašenja aktivnosti [3]



Slika . Životni ciklus aktivnosti [3]

## *Entity* radni okvir

*Entity* radni okvir (engl. *framework*) otvorenog je kôda s potporom Microsofta. Radni okvir omogućuje objektno-relacijsko mapiranje (engl. *object-relational mapping*, ORM), što u konačnici programerima olakšava rad s bazom podataka, svodeći rad na objektnu paradigmu. Drugim riječima, eliminira potrebu za većinom kôda za pristup podacima koji su programeri uglavnom morali ručno pisati[4]. Nadalje, *Entity* radni okvir rješava problem nekompatibilnosti tipova podataka baze podataka i .NET objekta korištenjem ORM alata. ORM alat mapira podatke iz baze u .NET objekte ili obratno i stvara naizgled virtualnu bazu podataka koja se koristi unutar samog kôda. Postoje različiti pristupi mapiranja:

* *EF Designer from database* pristup korištenjem unaprijed izrađene baze podataka generira konfiguracijsku *.edmx* datoteku koja može biti pregledana i mijenjana korištenjem vizualnog i interaktivnog EF dizajnera. Datoteka također generira i klase kojima se programer koristi, a odgovaraju entitetima iz baze podataka.
* *Empty EF Designer model* pristup generira praznu .*edmx* datoteku. Programer sam dizajnira model korištenjem EF dizajnera nakon čega se generiraju klase koje programer može koristiti u aplikaciji. Iz tog modela se može generirati skripta za kreiranje baze podataka.
* *Empty Code First model* pristup kreira prazan objekt modela kojeg programer može definirati isključivo putem kôda. Iz modela se može generirati skripta za kreiranje baze podataka.
* *Code First from database* pristup kreira model i klase korištenjem unaprijed izrađene baze podataka. Model se koristi za komuniciranje s bazom, dok klase služe za korištenje unutar aplikacije.

## Microsoft SQL Server

Microsoft SQL Server je sustav za upravljanje relacijskim bazama podataka. Njegova glavna komponenta je SQL *server database engine*, koja prati i kontrolira pohranu podataka, procesiranje podataka i sigurnost podataka. Microsoft SQL server koristi T-SQL upitni jezik koji je implementacija SQL upitnog jezika. Omogućuje pisanje procedura, grananja, lokalne varijable, procesiranje datuma i brojne druge funkcionalnosti.

Postavlja se, međutim, pitanje, na koji način SQL server kreira i izvršava upite. Pojednostavljeno, upitni procesor (engl. *query processor*) je komponenta SQL servera koja određuje korake kojima će se traženi podaci iz upita dohvatiti ili obraditi. Spomenuti niz koraka naziva se upitnim planom (engl. *query plan*). Pritom upitni procesor odrađuje i upitnu optimizaciju (engl. *query optimization*). Upitna optimizacija je vrlo važna jer jedan upit može imati više od jednog upitnog plana. Primjerice, za upit koji sadrži *join* naredbu za spajanje dviju tablicu i *select* naredbu za dohvaćanje podataka postoje dva upitna plana:

* izvršiti spajanje tablica, zatim dohvatiti podatke
* dohvatiti podatke iz obje tablice, zatim spojiti dohvaćene podatke

Upitni procesor, uzevši u obzir oba upitna plana, bira onaj koji će imati najkraće vrijeme izvršenja, odnosno optimalno rješenje. [9]

SQL Server omogućuje da više korisnika koristi bazu podataka u isto vrijeme. S obzirom na to da jedan korisnik može biti u procesu dohvaćanja podatka kojeg drugi korisnik u isto vrijeme mijenja, važan je integritet podataka. Rješenje SQL Servera za to je kontrola istodobnosti (engl. *concurrency control*). Kontrola istodobnosti ima dva načina čuvanja integriteta: pesimistična istodobnost i optimistična istodobnost. [9]

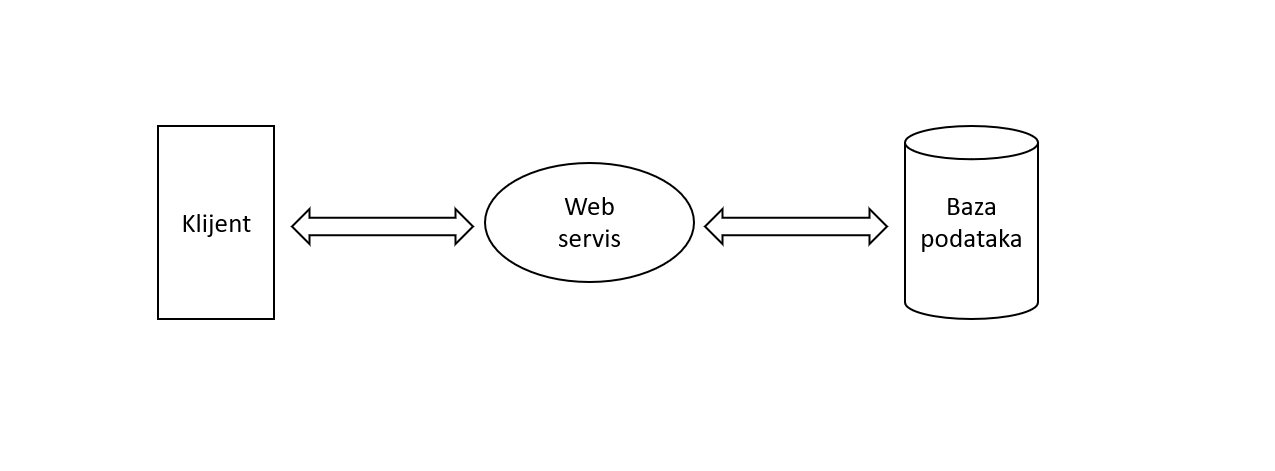
Pesimistična istodobnost koristi brave (engl. *lock*) koje mogu biti ekskluzivne (engl. *exclusive*) ili dijeljene (engl. *shared*). Ekskluzivne brave daju samo jednom korisniku pristup podacima, te ni jedan drugi korisnik nema pristup podacima dok god je brava aktivna. Dijeljena brava se koristi kod čitanja podataka. Više korisnika može čitati iste podatke, ali ni jedan korisnik ne može pritom dobiti ekskluzivnu bravu nad tip podacima. Ekskluzivna brava može se dobiti nad podacima tek kad su maknute sve dijeljene brave. [9]

Optimistična istodobnost omogućuje neometan pristup podacima svim korisnicima. Kada se podaci u jednom redu promijene, novi red se stvara s promijenjenim podacima, a stari se čuva. Svaki od njih može biti identificiran po transakcijskom jedinstvenom identifikatoru. Iako se oba reda čuvaju u bazi, stari red se sprema u sistemsku bazu podataka nazvanu *Tempdb*. Tijekom mijenjanja reda, ako dođe novi zahtjev nad tim redom, koristi se stara verzija reda za obradu, dok god zahtjev za promjenom podataka nije izvršen. [9]

# Arhitektura sustava

Sustav *RemoteTrainer* koristi troslojnu arhitekturu sustava. Slojevi sustava su:

* prezentacijski sloj koji je dio klijentskog sloja, izveden kroz Android mobilnu aplikaciju,
* sloj poslovne logike koji je dio poslužiteljskog sloja predstavlja web servis,
* podatkovni sloj koji je dio poslužiteljskog sloja predstavlja baza podataka.



Slika . Arhitektura sustava *RemoteTrainer*

Komunikacija u sustavu uglavnom je inicirana od strane klijenta prema bazi podataka, na načine opisane u nastavku.

Prvi način je slanje podataka. Podaci se od strane mobilne aplikacije šalju web servisu na obradu. Web servis potom vraća povratnu informaciju mobilnoj aplikaciji o statusu zahtjeva. Web servis podatke nakon obrade i provjere da su u ispravnom obliku šalje u bazu podataka na trajnu pohranu.

Drugi način je zahtjev za dohvat podataka. Zahtjev za podacima mobilna aplikacija šalje web servisu koji dohvaća potrebne podatke iz baze podataka, zatim ih obrađuje u oblik potreban i prilagođen klijentu. Ako su podaci uspješno dohvaćeni iz baze podataka i obrađeni, šalje ih nazad mobilnoj aplikaciji. U suprotnom, vraća se kôd koji označava iznimku, koja detaljno opisuje što je uzrok pogreške.

Kod ovakve arhitekture izuzetno je važna interoperabilnost svakog sloja. Interoperabilnost je svojstvo aplikacije da s drugim tipovima aplikacija ili servisa radi u cjelini kao jedan sustav. Interoperabilnost slojeva u sustavu *RemoteTrainer* postignuta je korištenjem JSON formata kao standarda za komunikaciju između slojeva. [10]

Ovakva arhitektura sustava omogućuje neovisnost svakog sloja. Klijentska strana izrađena je *nativno* za Android mobilne uređaje. Ako poželimo dodati novog klijenta koji je *nativan* za *iOS* mobilne uređaje, što predstavlja novu aplikaciju, ostali slojevi ne zahtijevaju nikakvu nadogradnju ili izmjenu, već novi klijentski sloj može odmah koristiti ostale slojeve za komunikaciju. [11]

Još jedna dobrobit ovako realiziranog sustava je horizontalno skaliranje. Naime, ako dođe do velike potražnje za aplikacijom diljem svijeta, može doći zagušenosti prometa u komunikaciji između slojeva. U tom slučaju, postoji potreba za dodatnim poslužiteljem koji bi se uklopio u arhitekturu klastera (engl. *cluster*). Pojednostavljeno, oba poslužitelja bi tada istovremeno mogla obavljati iste zadatke i time se međusobno rasterećivati. U slučaju *RemoteTrainer* sustava, dovoljno je zakupiti poslužitelj na Azure *cloud* servisu i postaviti web servis na njega. Dodatni poslužitelj tada bi preuzeo dio prometa s prvog i tako smanjio opterećenje. Horizontalno skaliranje može se primijeniti još i u slučaju sporih odgovora kod korisnika koji su daleko od mjesta poslužitelja. Novi poslužitelj se otvara na njima bližoj lokaciji i dobiva se na brzini.

## Poslužiteljski sloj

Poslužiteljski sloj sastoji se od web servisa i baze podataka. Poslužiteljski sloj nema korisničko sučelje i služi isključivo za obradu, slanje, primanje i spremanje podataka. Klijentski sloj mobilne aplikacije nema izravan pristup bazi podataka, već sva komunikacija ide preko web servisa. Poslužiteljski sloj sastoji se od aplikacijskog programskog sučelja i podatkovnog sloja.

API, odnosno aplikacijsko programsko sučelje, definira način interakcije između dva ili više programa te omogućuje modularno programiranje, odnosno dodavanje novih modula i pristupnih točaka koje programi mogu koristiti bez utjecanja na druge. *RemoteTrainer* sustav sadržava web aplikacijsko programsko sučelje koje koristi HTTP protokol za komunikaciju i JSON format kao definirani način slanja i primanja podataka između klijenta i web servisa.

Podatkovni sloj definira mjesto gdje se čuvaju podaci. To je najčešće baza podataka u većim sustavima, kao i u sustavu *RemoteTrainer*. Podatkovni sloj može još biti i tekst datoteka, ili *excel* tablica i drugo.

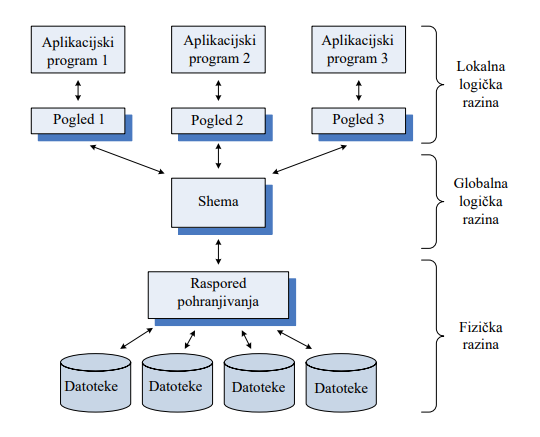
### Baza podataka

Baza podataka je skupina podataka organizirana kako bi se jednostavno moglo njima pristupati, upravljati i mijenjati. Podaci su dostupni raznim korisnicima i programima. CRUD operacije nad bazom provodi sustav za upravljanje bazom podataka (engl. *Database Management System*, DBMS).

DBMS oblikuje fizički prikaz baze podataka u skladu s traženom logičkom strukturom i poslužuje podatke. Obavlja sve operacije s podacima, brine za sigurnost, automatizira administrativne poslove i može podržati više vrsta baza podatka, svaka od njih može imati svoju logičku strukturu, ali u skladu s istim fizičkim prikazom. [13]

Arhitektura baze podataka (Slika 4.2) sastoji se od:

* Fizičkog sloja – određuje fizički prikaz i raspored podataka na vanjskim jedinicama, može se dalje dijeliti na više podrazina apstrakcije [13]
* Globalnog logičkog sloja – definira logičku strukturu baze u obliku sheme (engl. *schema*). Shema može biti tekst ili dijagram. Unutar sheme definiraju se svi tipovi podataka i veze između njih [13]
* Lokalnog logičkog sloja – definira dio baze kojim se koristi pojedina aplikacija. Definira ga pogled (engl. *view*). To je tekst ili dijagram kojim se definiraju lokalni tipovi podataka i veze između njih. Također, pogled definira način na koji se iz globalnih podataka i veza izvode lokalni. [13]



Slika . Arhitektura baze podataka [13]

Nadalje, razlikujemo različite tipove baze podataka:

* Relacijske – baze podataka u kojima su podaci organizirani u tablice koje se sastoje kolona i redaka [12]
* Objektno orijentirane – podaci su spremljeni u obliku objekata kao u objektno-orijentiranom programiranju [12]
* Raspodijeljene – podaci u bazi se spremaju na više računala na istoj fizičkoj lokaciji ili na više računala iz drugih mreža [12]
* Skladišta podataka – baze podataka dizajnirane za brze upite i analizu [12]
* NoSQL – ne relacijske baze podataka koje dopuštaju nestrukturirano ili polu-strukturirano čuvanje podataka [12]
* OLTP – brza analitička baza podataka dizajnirana za veliku količinu transakcija proizvedenih od više korisnika [12]

Baza podataka, kao krajnji dio aplikacije, služi isključivo za spremanje i dohvat podataka. Svaka tablica unutar baze podataka predstavlja jedan entitet. Istovremeno, svaki entitet ima redove i kolone. Redovi su predstavljeni jedinstvenim identifikatorom kojim se ujedno i pristupa točno određenom redu u tablici. Jedna kolona predstavlja jedan podatak o tom entitetu. Svaka kolona ima svoj tip podataka koji ujedno služi i kao restrikcija radi osiguravanja sigurnosti podataka.

CRUD operacije su nužne za implementiranje dosljedne (engl. *persistent*)baze podataka. Kreiranje (engl. *create*) omogućuje korisnicima dodavanje novog zapisa u tablicu. Ova operacija koristi *insert* naredbu u upitnom jeziku. Čitanje (engl. *read*)omogućuje čitanje određenih podataka. Među ostalim, ova operacija uglavnom koristi *select* i *join* naredbe. Ažuriranje (engl. *update*) omogućuje promjenu podataka unutar jednog reda. Operacija koristi *update* naredbu. Brisanje (engl. *delete)* omogućuje micanje reda iz tablice. Baze podataka mogu omogućiti jako (engl. *hard*)ili lagano (engl. *lagano*)brisanje. Jako brisanje miče red iz tablice, dok lagano brisanje miče indikator da je red u tablici, a podatke čuva, odnosno arhivira.

Svaka operacija mora zadovoljavati ACID svojstva podataka. ACID osigurava ispravnost podataka u slučaju pogrešaka računala, gubitka električne struje i ostalih nepogoda. [14] Definiraju ga četiri svojstva:

* Valentnost (en*g*l. *atomicity*) – garantira da je svaka naredba u upitu jedna jedinica. Svaka jedinica nakon izvršenja vraća rezultat uspješnosti. Ako je jedna jedinica vratila pogrešku, cijela transakcija vraća pogrešku [14]
* Konzistentnost (engl. *consistency*) – osigurava da transakcija šalje podatke iz jednog ispravnost stanja u drugi, prateći sve restrikcije definirane na bazi podataka [14]
* Izolacija (engl. *isolation*)– osigurava da istovremeno izvršavanje transakcija ostavi bazu podataka u istom stanju kao da su se transakcije izvršavale jedna nakon druge [14]
* Trajnost (engl. *durability*) – garantira da nakon što se transakcija potvrdila, ostat će potvrđena čak i ako sistem padne ili se ugasi [14]

Kako bi osigurali konzistentnost podataka te da su podaci na pravom mjestu unutar logičke sheme baze podataka provodi se normalizacija. Normalizacija je organiziranje entiteta u svrhu dobivanja učinkovite i fleksibilne baze podataka. Postoje više forma normalizacije, od kojih su prve tri korištene u *RemoteTrainer* sustavu:

* Prva normalna forma – svi entiteti imaju jedinstveni ključ koji se može sastojati od jednog ili više atributa. Svaki atribut mora biti jedinstven unutar entiteta [13]
* Druga normalna forma – svi atributi koji nisu dio jedinstvenog identifikatora moraju ovisiti samo o njemu [13]
* Treća normalna forma – svi atributi koji nisu dio jedinstvenog identifikatora ne smiju biti međusobno ovisni [13]

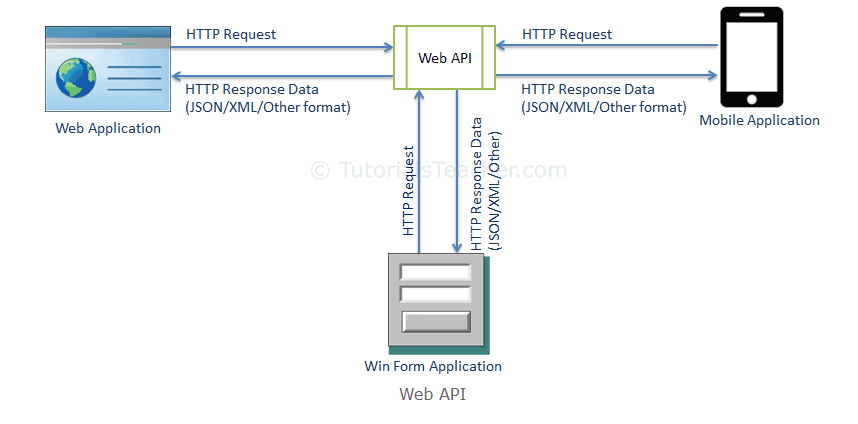
### Web servis

Web servis u sustavu *RemoteTrainer* realiziran je s ASP.NET Web API radnim okvirom. Okvir omogućuje izradu HTTP servisa kojem može pristupati velik broj korisnika i programa. Koristi HTTP protokol i HTTP/RPC pristup za komunikaciju. Takav pristup omogućava slanje HTTP zahtjeva koji zatim pozivaju metode. [14] Postoje četiri osnovna zahtjeva koji se ujedno koriste u *RemoteTrainer* sustavu:

* GET – zahtjev za dohvaćanje
* POST – zahtjev s tijelom za slanje podataka
* PUT – zahtjev s tijelom za slanje ili izmjenu podataka
* DELETE – zahtjev za brisanje podataka

Nakon zahtjeva, poslužitelj redovito vraća kod koji govori o uspješnosti zahtjeva. Općenito značenje kôda definira prvi broj u nizu:

* 1xx – zahtjev je primljen
* 2xx – zahtjev je uspješan
* 3xx – zahtjev zahtjeva dodatne akcije kako bi bio uspješan
* 4xx – zahtjev ne može biti procesiran
* 5xx – server nije uspio ispuniti zahtjev



Slika . Komunikacija web servisa [15]

Web servis je središnji dio aplikacije. Komunicira s mobilnom aplikacijom kao klijentom i bazom podataka. Njegova je zadaća da pazi da su podaci ispravni, pravilnog oblika i prihvatljivi za spremanje. Nadalje, web servis obrađuje i prilagođuje podatke u oblik koji klijentu odgovara.

S druge strane, komunikacija s klijentom je napisana od strane programera. Podaci koji se primaju i šalju od klijenta do web servisa i obratno su točno određenih modela koji u većini slučajeva ne odgovaraju jedan naprema jedan prema entitetu iz baze podataka, upravo zato što se nad bazom provodi normalizacija. Na primjer, u bazi se podaci ime, prezime, dokumenti, slika, adresa, mjesto prebivališta, broj telefona nekog korisnika čuvaju u više entiteta unutar baze podataka, dok se svi ti podaci nalaze u klasi *UserProfile* klijentske aplikacije. Iz tog razloga se arhitektura web servisa dijeli na komuniciranje s klijentom putem *RequestModel* i *ResponseModel* klasa,.

Klijent podacima pristupa ili šalje zahtjev za podacima preko API krajnjih točaka. API krajnje točke su točno određena URL adresa kojoj klijent pristupa. Prilikom pristupa adresi web servis pokreće povezanu metodu s tom adresom i obrađuje zahtjev.

## Klijentski sloj

Klijentski sloj predstavlja sloj kojim korisnik direktno koristi, komunicira i manipulira podacima u sustavu. U sustavu *RemoteTrainer*, klijentski sloj je realiziran Android mobilnom aplikacijom.

Ugrubo, postoje dvije vrste prezentacijskog sloja:

* tanki klijent (engl. *thin client*) – oni se bave gotovo isključivo prikazom podataka, dok se sve kalkulacije i obrade podataka događaju na centraliziranom poslužitelju
* debeli klijent (engl. *fat client*) – nisu ovisni o poslužiteljskoj aplikaciji, mogu se koristiti i kada nema mreže

Tablica . Usporedba tankog i debelog klijenta [16]

|  |  |
| --- | --- |
| Tanki klijent | Debeli klijent |
| Jednostavno postavljanje aplikacije | Teži za postavljanje, trebaju posebne softvere. |
| Ako poslužitelj padne, klijent ne može nastaviti s radom | Klijent može raditi i u slučaju pada poslužitelja |
| Niži sigurnosni rizik | Veći sigurnosni rizik jer se dio obrade vrši na klijentu |
| Portabilnost preko više vrsta uređaja | Smanjena portabilnost |
| Veće opterećenje poslužitelja | Manje opterećenje poslužitelja |

U sustavu *RemoteTrainer* koristi se debeli klijent. Aplikaciji se može pristupiti kada nema mreže, međutim neke funkcionalnosti neće biti dostupne.

Kao što je već navedeno, klijentski sloj omogućava slanje zahtjeva za određenim podacima putem API točaka, kao i primanje podataka. Prilikom slanja podataka, klijentski sloj ne provjerava nužno ispravnost podataka, već samo onemogućuje unos neispravnih oblika podataka. Validacija ispravnosti podataka je odgovornost dodijeljena poslužiteljskom sloju.

Klijenata teoretski može biti neograničen broj. Svi će pristupati istom web servisu i dobivati podatke iz zajedničke baze podataka.

### Komunikacija aplikacije s web servisom

Komunikacija aplikacije s web servisom je uglavnom inicirana od strane klijentskog sloja prema web servisu. Riječ je o *duplex* komunikaciji, odnosno podaci se šalju u oba smjera. Klijentski sloj šalje podatke ili zahtjev za podacima web servisu. Podaci koji se izmjenjuju su realizirani u JSON formatu za spremanje podataka. Svaki podatak predstavlja jedan model koji odgovara jednom objektu ili listi objekata. Istovremeno, objekti su točno određenih i točnih formata.

# Implementacija programskog rješenja

Programsko rješenje sastoji se od tri djela i svaki je implementiran koristeći različite tehnologije. Klijentski dio i web servis komuniciraju JSON formatom. JSON format je lako čitljiv ljudima i računalima. Sadrži parove sastavljene od ključa i vrijednosti. Vrijednost može sadržati svoje parove ključa i vrijednosti što u cjelini čini objekt. Također, može definirati listu parova.

Komunikacija između web servisa i baze odrađena je *entity framework* tehnologijom za koju je potreban konfiguracijski podatak nazvan *connectionString*. On se koristi kako bi aplikacija mogla identificirati server i bazu podataka na koji se treba spojiti, kao i podatke za autorizaciju. Sprema se najčešće u konfiguracijskog datoteci u aplikaciji koja želi pristupiti bazi podataka.

S obzirom na navedeno, svaki dio će biti detaljnije objašnjen u ovom poglavlju.

## Servisni dio sustava

Servisni dio sustava dijelimo na dva djela: web servis i bazu podataka. On nema korisničko sučelje jer nema potrebe za istim. Načelno je jedan, ali koristi ga više od jednog klijenta.

Servisni dio postavljen je na Azure *cloud* servis koji omogućuje korištenje aplikacije kao uslugu. Azure *cloud* servis rješava problem vlastite računalne infrastrukture i omogućuje jednostavno dodavanje ili brisanje usluga. Usluge se uređuju i pristupaju web preglednikom. Azure oblak osigurava dostupnost usluga koje su postavljene na jednu od njihovih infrastruktura diljem svijeta.

### Baza podataka

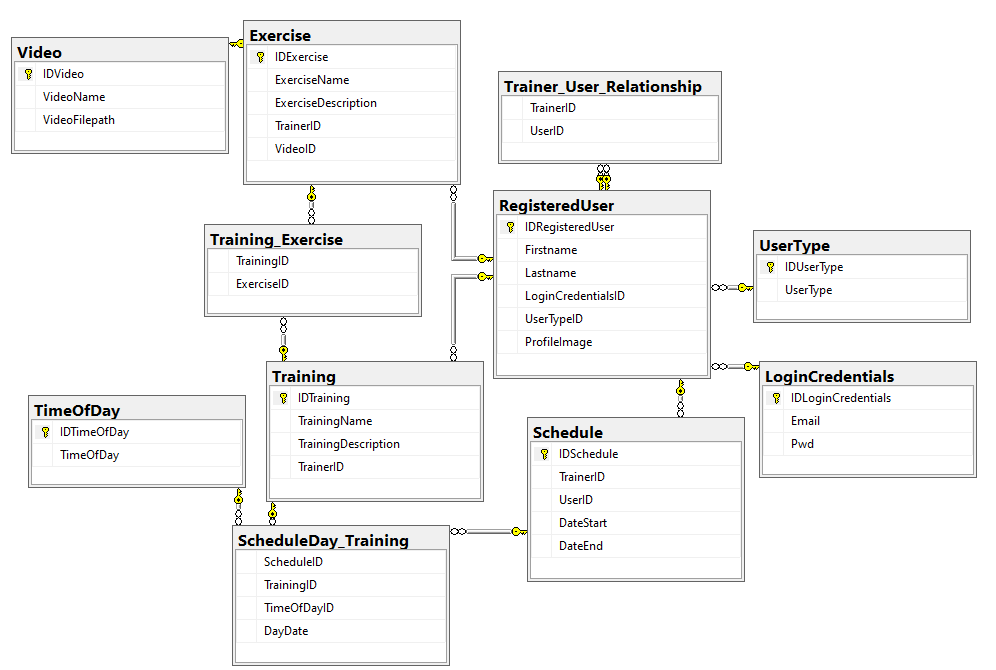
Baza podataka napravljena je korištenjem Microsoft SQL Server sustava, T-SQL upitnim jezikom. Komunikacija ide isključivo prema web servisu.

Kao što je već navedeno, naredbe baze podataka se nikada direktno ne zovu ni iz jednog sloja aplikacije, već ih *Entity* radni okvir poziva implicitno. Konkretnije, *DbContext* klasa koju nasljeđuju svi modeli generirani tehnologijom odrađuje CRUD operacije nad bazom podataka. Klasa sadrži javne metode koje koriste .NET objekte i koji moraju striktno odgovarati entitetu iz baze nad kojim se radi jedna od CRUD operacija. Zato je najbolje koristiti isključivo klase generirane od strane radnog okvira. Dohvaćanje podataka može se odraditi pozivanjem *Find()* ili *Where()* metoda. Za dodavanje reda u tablicu baze podataka koristi se *Add()* metoda. Za izmjenu podataka prvo se moraju dohvatiti podaci, zatim prilikom promjene jednog od podatka *DbContext* postavlja status objekta kao *EntityState.Modified* te nakon pozivanja funkcije *SaveChanged*, radni okvir obavještava bazu podataka o promjeni. Brisanje podataka dostupno je *Remove()* metodom. Nakon poziva jedne od navedenih metoda, radni okvir generira upit koji odgovara zahtjevu i šalje ga bazi podataka.

#### Model baze podataka

Model baze podataka čine sljedeći entiteti:

* *RegisteredUser* – glavni entitet modela, predstavlja registriranog korisnika. Registrirani korisnik sadrži informacije o imenu i prezimenu osobe, njegovu profilnu sliku i reference na *LoginCredentials* i *UserType* entitete.
* *LoginCredentials* – entitet koji čuva email i lozinku registriranog korisnika.
* *UserType* – entitet koji sadrži tip korisnika. Tip korisnika može biti trener ili korisnik usluga.
* *Trainer\_User\_Relationship* – vezna tablica kojoj je jedinstveni identifikator skup od dvije reference na entitet *RegisteredUser*. Navedeno omogućuje da se dva korisnika mogu pojaviti samo jedanput u istom vezi unutar ovog entiteta. Drugim riječima, jedan trener može imati samo jedan aktivan poslovni odnos s korisnikom.
* *Schedule* – entitet koji predstavlja raspored. Sadrži referencu na trenera koji je napravio raspored i korisnika za koga je napravljen raspored, datum od kada je aktivan i datum do kada je aktivan raspored.
* *ScheduleDay\_Training* – tablica koja čuva podatke o tome koji trening spada na koji dan, u koje doba dana i kojem rasporedu.
* *Training* – entitet jednog treninga. Sadrži ime i opis treninga, te referencu na trenera koji je napravio taj trening. Razlog iz kojeg svaki trening ima referencu na trenera je što se treninzi rade zasebno van rasporeda i mogu se pridjeljivati različitim rasporedima i različitim korisnicima.
* *TimeOfDay* – Tablica koja čuva moguća doba u danu koje se može jednom danu u rasporedu.
* *Training\_Exercise* - tablica kojoj je jedinstveni identifikator skup od dvije reference na entitete *Training* i *Exercise*. Svaki trening može imati više vježbi, ali i svaka vježba može biti u više treninga, stoga najbolje rješenje je dvostruko vezana tablica.
* *Exercise* – entitet koji predstavlja jednu vježbu. Svaka vježba sadrži ime, opis, referencu na *Video* i referencu na trenera. Svaka vježba sadrži referencu na trenera jer se vježbe u aplikaciji rade neovisno o treninzima i rasporedima. Time se vježbe i treninzi mogu ponovno upotrebljavati.
* *Video* – entitet jednog videa koji služi za pohranu imena videa i URL koji će se koristiti za prikazivanje videa.

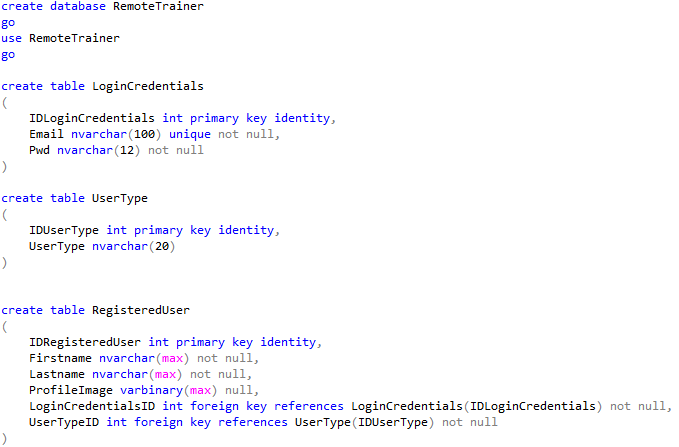


Slika . Relacijski model baze podataka

#### Implementacija baze podataka

Baza podataka je implementirana u Microsoft SQL Server sustavu, korištenjem T-SQL upitnog jezika.

Baza se kreira pokretanjem skripte *create\_database.sql* koja je kreirana s pažnjom prema relacijama između entiteta. Drugim riječima, ako entitet sadrži referencu prema drugom entitetu u obliku stranog ključa, prvo se mora kreirati referencirani entitet. Primjerice, prvi entiteti koji se kreiraju nakon pokretanja skripte su *LoginCredentials* i *UserType*. Nakon njih, može se kreirati entitet *RegisteredUser* koji referencira spomenute entitete. Potom se kreiraju ostali entiteti koji u pravilu referenciraju *RegisteredUser* entitet.



Slika . Prikaz kreiranja prva tri entiteta

Tijekom implementacije postojala je mogućnost za korištenjem procedura za dohvat podataka, ali s obzirom na spektar mogućnosti *Entity* radnog okvira koji većinu odradi sam, nije bilo potrebe za procedurama.

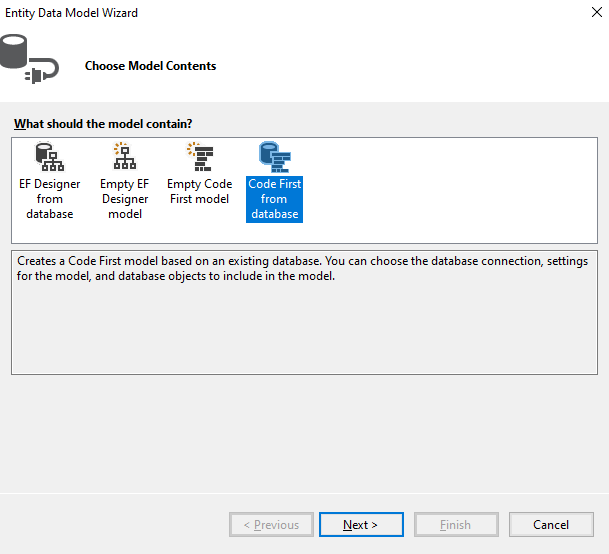
### Web servis

Web servis implementiran je korištenjem *Visual Studio* integriranog razvojnog okruženja, u .NET radnoj okolini, uz pomoć WEB APItehnologijeu programskom jeziku C#.

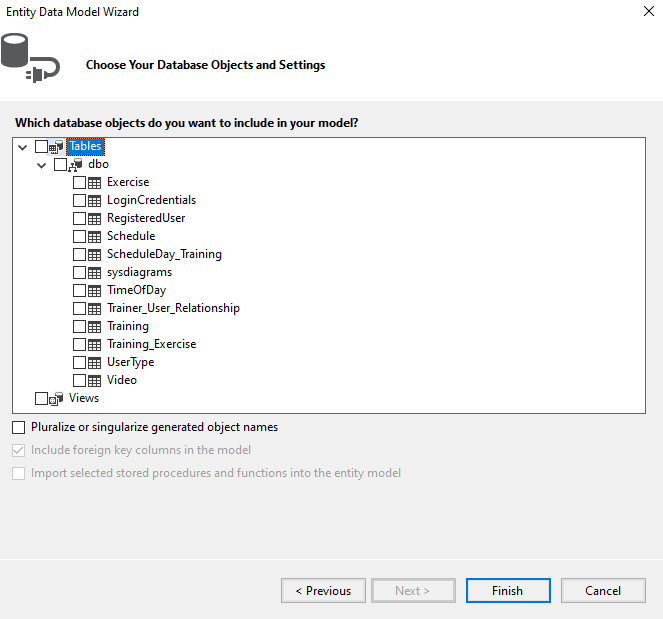
Komunikacija prema klijentu koristi API pristupne točke, te *ResponseModel* i *RequestModel* objekte za primanje i slanje objekta točnog određenog oblika.

#### Implementacija podatkovnog modela

Implementacija podatkovnog modela je rađena na *Code first from database* principu putem *Entity* radnog okvira korištenjem *ADO.NET Entity Data Modela* koji generira sve potrebne objekte i model za pristup bazi podataka. Model se kreira tako da se odabere željeni pristup bazi (Slika 5.3), određene postavke modela i *connectionString*, a zatim tablice koje će model koristiti (Slika 5.4).



Slika . Prikaz izbornika za odabir način kreiranja modela



Slika . Prikaz izbornika za odabir tablica tijekom kreiranja modela

Nakon što je model kreiran, bazi se pristupa pomoću klasa kreiranih uz taj model. Kreirane objekti odgovaraju jedan naprema jedan entitetu iz baze.

Na primjer, ako smo kreirali model *RegisteredUserModel* s objektima *RegisteredUser, LoginCredentials, UserType*. Model sadržava sljedeći kôd:

public partial class RegisteredUserModel : DbContext

{

public RegisteredUserModel(): base("name=webCS"){}

public virtual DbSet<LoginCredentials> LoginCredentials {get; set; }

public virtual DbSet<RegisteredUser> RegisteredUser { get; set; }

public virtual DbSet<UserType> UserType { get; set; }

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Entity<LoginCredentials>()

.HasMany(e => e.RegisteredUser)

.WithRequired(e => e.LoginCredentials)

.HasForeignKey(e => e.LoginCredentialsID)

.WillCascadeOnDelete(false);

modelBuilder.Entity<UserType>()

.HasMany(e => e.RegisteredUser)

.WithRequired(e => e.UserType)

.HasForeignKey(e => e.UserTypeID)

.WillCascadeOnDelete(false);

}

}

Kôd 5.1 Prikaz kôda nakon kreiranja modela

Nakon kreiranja modela na sljedeći način možemo dobiti korisnika kojemu odgovaraju email i lozinka:

RegisteredUser result = registeredUserModel.RegisteredUser.Where(u => u.LoginCredentials.Email == loginCredentials.Email && u.LoginCredentials.Pwd == loginCredentials.Pwd).FirstOrDefault();

#### Implementacija pristupnih točaka

Objekt u kojemu se definiraju API pristupne točke se zove kontroler (engl. *Controller)*. Svaki kontroler nasljeđuje *ApiController* objekt. (Kôd 5.2) prikazuje API pristupnu točku za autorizaciju korisnika u *LoginCredentialsController* kontroleru.

Dio kôda [Route("api/login")] definira putanju do ove API točke. Pozivanjem te putanje s pravilnim oblikom tijela (engl. *body*). Potom se kroz model pokušava dobiti korisnik koji ima jednak email i lozinku koji su poslani u tijelu *HttpPost* zahtjeva u obliku *LoginCredentials* objekta. Ako nije pronađen, metoda vraća *NotFound()*. Ako je pronađen, kreira se novi objekt *RegisteredUserResponseModel*. Taj objekt sadrži točno one podatke koje klijenta zanimaju u trenutku autorizacije.

private RegisteredUserModel registeredUserModel = new RegisteredUserModel();

[Route("api/login")]

public async Task<Object> Login(Models.LoginCredentialsClasses.LoginCredentials loginCredentials)

{

RegisteredUser result = registeredUserModel.RegisteredUser.Where(u => u.LoginCredentials.Email == loginCredentials.Email && u.LoginCredentials.Pwd == loginCredentials.Pwd).FirstOrDefault();

if (result == null)

{

return NotFound();

}

RegisteredUserResponseModel userModel = new RegisteredUserResponseModel

{

IDUser = result.IDRegisteredUser,

FirstName = result.Firstname,

LastName = result.Lastname,

Email = result.LoginCredentials.Email,

Type = result.UserTypeID

};

return Ok(userModel);

}

Kôd 5.2 API pristupna točka

#### Komunikacija web servisa s bazom podataka

Komunikacija je odrađena putem *Entity* radnog okvira. Koriste se modeli za dohvaćanje, spremanje, ažuriranje i brisanje podataka iz baze. Web servis je zaslužan za provjeru oblika podatka koji se šalje u bazu. Također, web servis je zaslužan za obradu podatka nakon dohvaćanja iz baze podataka.

[Route("api/register")]

[ResponseType(typeof(RegisteredUser))]

public IHttpActionResult PostRegisteredUser(RegisteredUserRequestModel registeredUser)

{

try

{

var user = new RegisteredUser

{

Firstname = registeredUser.FirstName,

Lastname = registeredUser.LastName,

LoginCredentials = new LoginCredentials

{

Email = registeredUser.Email,

Pwd = registeredUser.Password

},

UserTypeID = registeredUser.UserTypeID

};

db.RegisteredUser.Add(user);

db.SaveChanges();

}

catch (DbUpdateException e)

when (e.InnerException?.InnerException is SqlException sqlEx && (sqlEx.Number == 2601 || sqlEx.Number == 2627))

{

string message = "";

string data = sqlEx.Message.Split('(', ')')[1];

if (data == registeredUser.Email)

return Conflict();

return InternalServerError();

}

catch (Exception ex)

{

return InternalServerError();

}

return Ok();

}

Kôd 5.3 Registriranje korisnika i dodavanje korisnika u bazu podataka

## Klijentski dio sustava

Klijentski dio sustava sastoji se od android mobilne aplikacije. Aplikacija služi za povezivanje dva tipa korisnika u svrhu obavljanja poslovne suradnje uz pomoć alata dostupnih putem *RemoteTrainer* aplikacije. Korisnike predstavljaju trener i korisnik usluga trenera. Putem aplikacije treneri mogu pregledavati svoje korisnike, zahtjeve za poslovne odnose, sastavljati vježbe, treninge i rasporede svojim korisnicima. Korisnici mogu poslati treneru zahtjev za suradnju, platiti trenerove usluge, pregledavati svoj raspored, treninge i vježbe.

### Funkcionalnosti mobilne aplikacije

U ovom poglavlju će detaljnije biti objašnjene glavne funkcionalnosti mobilne aplikacije. Sve funkcionalnosti lako su uporabljive preko izbornika. Izbornik (Slika 5.7) je napravljen s *NavigationView* komponentom koja definira svoj izgled *xml* datotekom. *NavigationView* komponenta također definira izgled zaglavlja (engl. *header layout*) i menija (engl. *menu items*). *header\_layout* *xml* datoteka definira izgled vrha izbornika (Slika 5.5). *menu\_items.xml* datoteka definira popis opcija prikazani na izborniku (Slika 5.6). S obzirom na dvije vrste korisnika, oni nemaju iste ekrane i funkcionalnosti, postoje dvije *menu items* datoteke, jedna za korisnika i jedna za trenera.

<?xml version="1.0" encoding="utf-8"?>

<menu xmlns:android="http://schemas.android.com/apk/res/android">

<item

android:id="@+id/miTrainerSearch"

android:title="@string/findTrainerString"></item>

<item

android:id="@+id/miISchedule"

android:title="@string/scheduleStringLowerCase"></item>

<item

android:id="@+id/miProfile"

android:title="@string/profileString"></item>

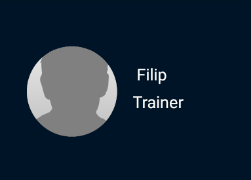
<item

android:id="@+id/miSettings"

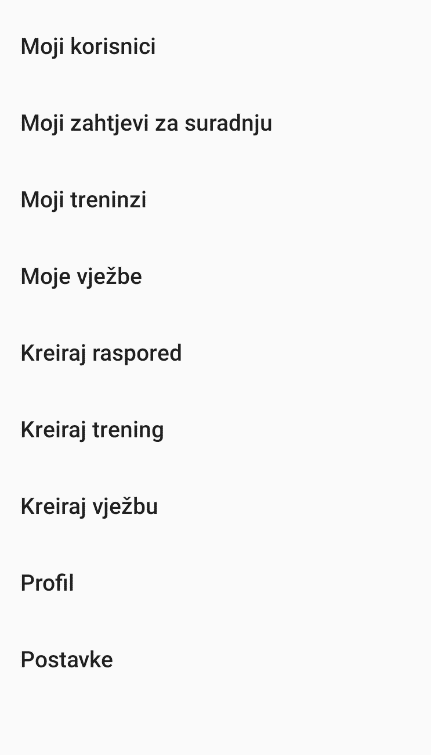
android:title="@string/settingsString"></item>

</menu>

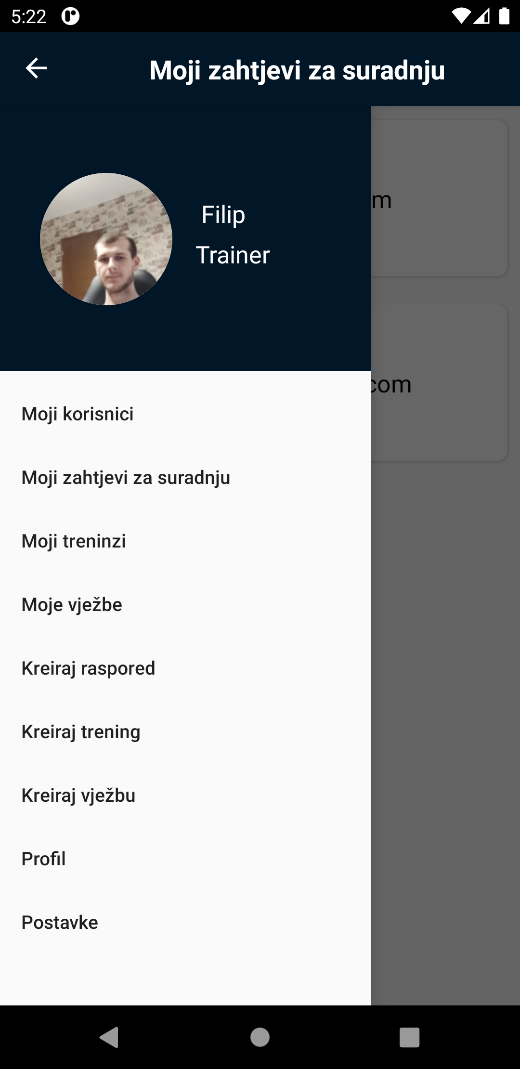
Kôd 5.4 *menu\_items.xml* za korisnika



Slika . Izgled zaglavlja izbornika



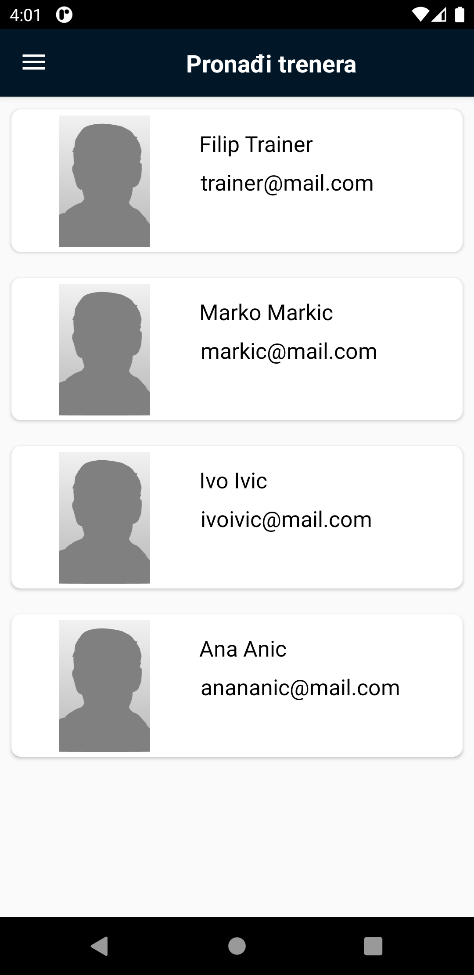
Slika . Izgled popisa opcija izbornika



Slika . Izgled izbornika za trenera

#### Pregled i dostupnosti trenera

Korisnik odabirom stavke „pronađi trenera“ iz izbornika odlazi na ekran (Slika 5.8) gdje su prikazani svi dostupni treneri. Treneri su ispisani u listi korištenjem *RecyclerView* komponente (Kôd 5.3). Prilikom klika na jednog od trenera se otvara njegov profil s osobnim informacijama.



Slika . Izgled ekrana pronađi trenera

MutableLiveData<List<Trainer>> trainers = ApiRequester.getTrainers();

trainers.observe(this, new Observer<List<Trainer>>() {

@Override

public void onChanged(List<Trainer> trainers) {

if(trainers == null)

{

return;

}

adapter = new TrainerPreviewAdapter(TrainerSearchActivity.this,

trainers);

rvTrainers.setAdapter(adapter);

}

});

Kôd 5.5 Prikaz svih trenera s *RecyclerView*  komponentom

(Kôd 5.3) prikazuje postavljanje *adaptera* u *RecyclerView* nakon što je web servis vratio rezultate i poslovni sloj aplikacije potvrdio da je rezultat valjan (Kôd 5.4). Poslovni sloj šalje *HttpGet* zahtjev za dohvaćanje trenera. Web servis vraća rezultat (Kôd 5.7). Nakon što poslovni sloj provjeri da je rezultat uspješno vraćen i ispravan, podaci se spremaju u *MutableLiveData* objekt. *MutableLiveData* objekt omogućuje asinkronu komunikaciju s web servisom tako što se aktivnost pretplati na metodu *onChanged* od *MutableLiveData* objekta (Kôd 5.3).

public static MutableLiveData<List<Trainer>> getTrainers(){

MutableLiveData<List<Trainer>> trainers = new MutableLiveData<>();

WebServiceAPI webServiceAPI =

getRetrofit().create(WebServiceAPI.class);

Call<List<Trainer>> call = webServiceAPI.getTrainers();

call.enqueue(new Callback<List<Trainer>>() {

@Override

public void onResponse(Call<List<Trainer>> call,

Response<List<Trainer>> response) {

if(response.isSuccessful() && response.body() != null)

{

trainers.setValue(response.body());

}

else{

trainers.setValue(null);

}

}

@Override

public void onFailure(Call<List<Trainer>> call, Throwable t

{

trainers.setValue(null);

}

});

return trainers;

}

Kôd 5.6 Zahtjev poslovnoj sloja za dohvaćanje trenera.

[Route("api/getTrainers")]

public List<RegisteredUserResponseModel> GetTrainers()

{

var users = db.RegisteredUser

.Where(user => user.UserTypeID == 2);

List<RegisteredUserResponseModel> trainers = new List<RegisteredUserResponseModel>();

foreach(RegisteredUser user in users)

{

RegisteredUserResponseModel trainer = new RegisteredUserResponseModel

{

IDUser = user.IDRegisteredUser,

FirstName = user.Firstname,

LastName = user.Lastname,

Email = user.LoginCredentials.Email

};

trainers.Add(trainer);

}

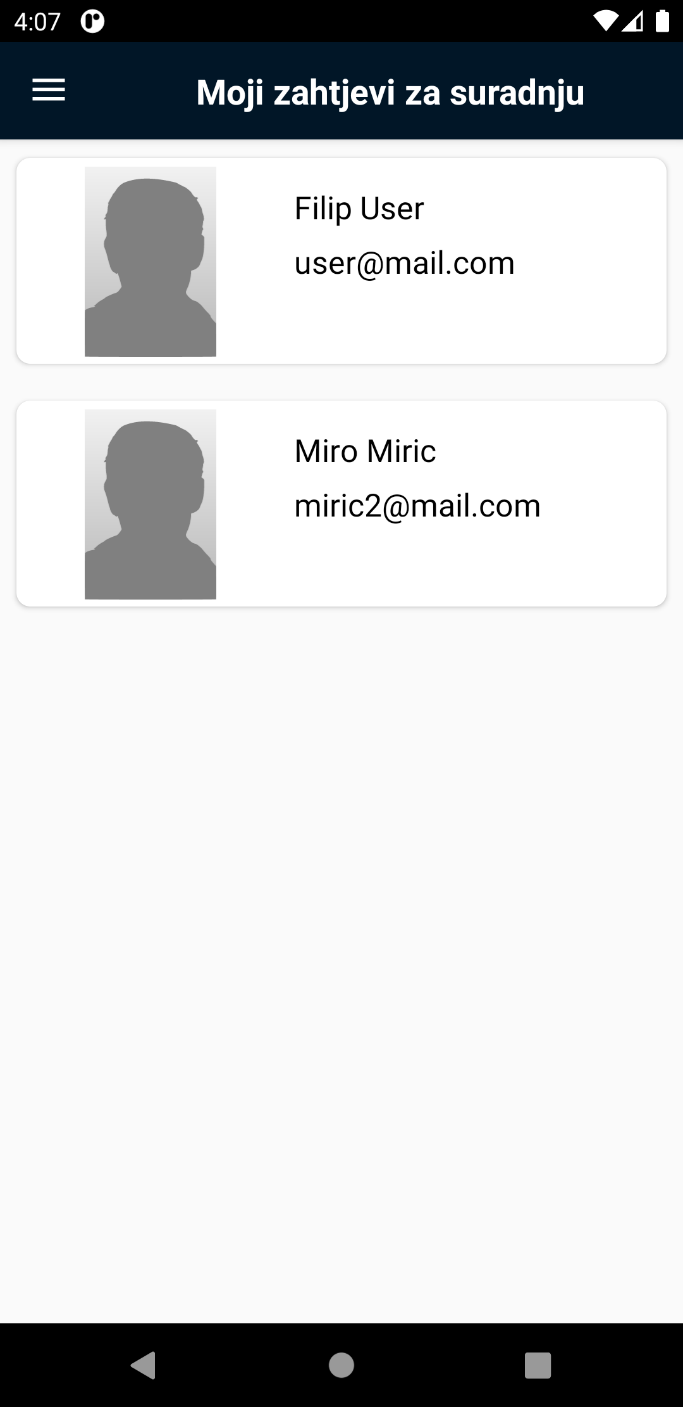
return trainers;

}

Kôd 5.7 Web servis metoda za dohvaćanje svih trenera

#### Najam trenera

Nakon što se korisniku otvori profil trenera prikazan je i gumb s nazivom „Pošalji zahtjev za suradnju“. Nakon klika na gumb web servisu se šalje jedinstveni identifikatori korisnika i trenera na obradu. Treneru se pojavljuje novi korisnik pod stavkom „Moji zahtjevi za suradnju“.

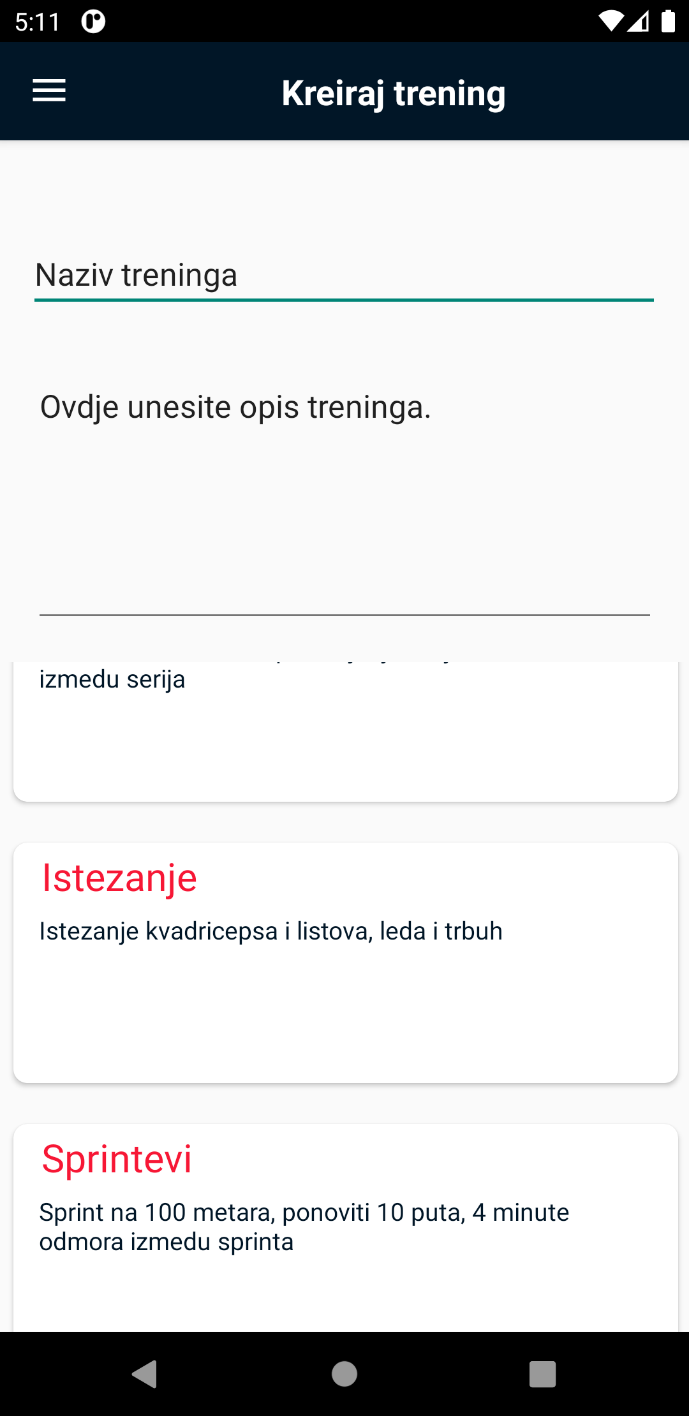


Slika . Ekran trenera s korisnicima koji su poslali zahtjev za poslovnu suradnju treneru

Trener potom može kliknuti na korisnika i pogledati njegov profil. Ako trener odobri zahtjev, korisnik i trener ulaze u poslovnu suradnju i trener može slagati raspored za korisnika. Ako trener odbije zahtjev, zahtjev se briše iz baze podataka.

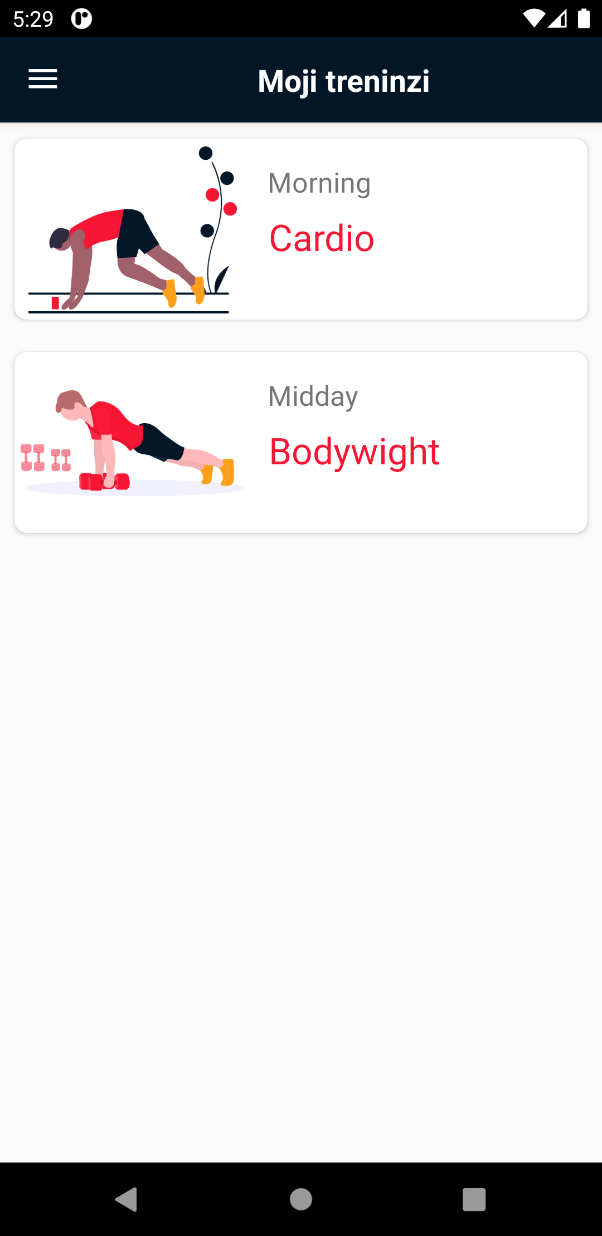
#### Sastavljanje treninga

Trener u bilo kojem trenutku može kreirati novi trening. Za kreiranje treninga potrebno je unijeti ime treninga, opis treninga i vježbe koje se izvode. Vježbe moraju unaprijed biti kreirane, te se klikom na vježbu označava koja je vježba aktivna za trenutni trening.



Slika . Ekran trenera za kreiranje treninga

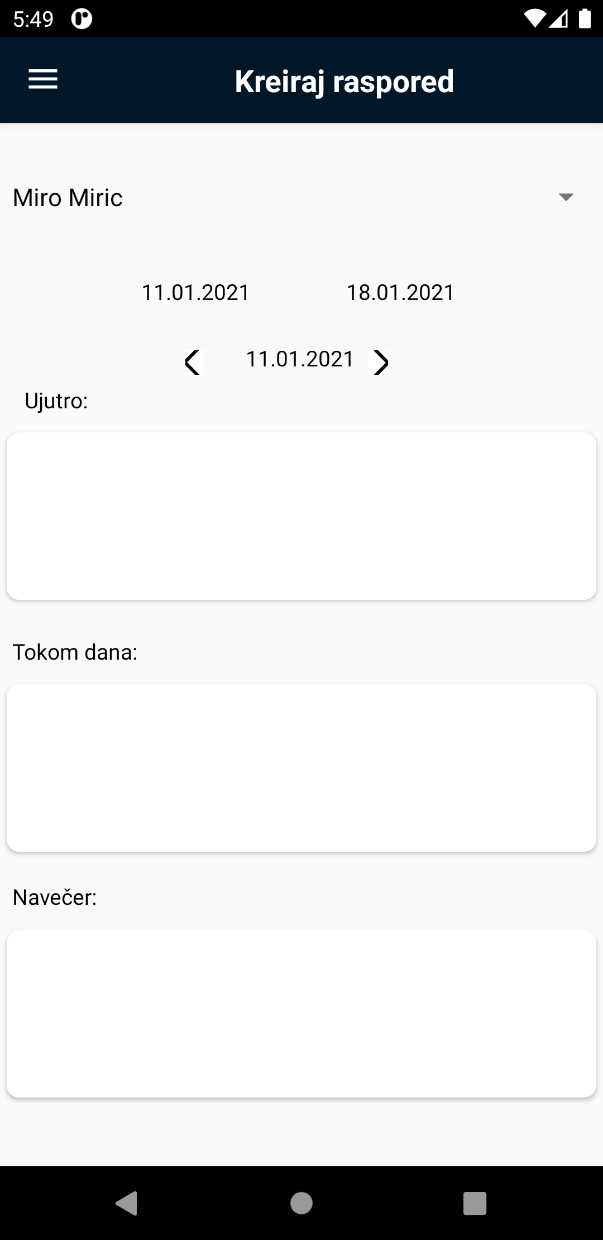
Trener može sve svoje treninge pregledati i izmijeniti odlaskom na stavku „Moji treninzi“ (Slika 5.11). Prilikom spremanja treninga, web servisu se šalje zahtjev s *RequestModel* objektom u tijelu *HttpPost* zahtjeva. Poslani objekt ne odgovara entitetima u bazi podataka, već ih web servis obrađuje i priprema za spremanje u bazu podataka.



Slika . Ekran trenera za pregled vlastitih izrađenih treninga

#### Sastavljanje rasporeda izvođenja

Trener može u bilo kojem trenutku sastaviti raspored za bilo kojeg korisnika s kojim ima sklopljenu poslovnu suradnju. Odlaskom na stavku „Sastavi raspored“ otvara se ekran za sastavljanje rasporeda.



Slika . Ekran trenera za kreiranje rasporeda

Trener prvo odabire korisnika, zatim datume od kada i do kada je raspored aktivan. Potom za svaki dan u rasporedu trener slaže unaprijed izrađene treninge u određeno doba dana. Trening se odabirem klikom na prazno mjesto, nakon čega se otvara ekran sa svim treninzima (Slika 5.11). Kada klikne na trening prazno mjesto se popuni s odabranim treningom. Prilikom spremanja raspored se šalje web servisu kao *RequestModel* na obradu.

#### Realizacija plaćanja korištenjem In-App Bilinga

Plaćanje se vrši od strane korisnika prema treneru za njegove usluge. Plaćanje je implementirano putem javnih PayPal API pristupnih točaka. Korisnik u dogovorenim terminima s trenerom, odlazi na trenerov profil, odabire opciju plaćanje, nakon čeka se otvara PayPal korisničko sučelje koje vrši transakciju. Automatski su unesena polja uplatitelja kao email adresa s kojom je korisnik prijavljen i primatelja kao email adresa trenera s kojom se registrirao.

### Opis korisnika

Aplikaciju koriste dva tipa korisnika, svaki tip ima posebne ekrane i funkcionalnosti. U ovom poglavlju su detaljnije opisani tipovi korisnika.

#### Trener

Trener je korisnik aplikacije koji pruža svoje usluge. Ima pristup funkcionalnostima kao što su:

* Pregled vlastitog profila
* Pregled svojih korisnika
* Pregled zahtjeva korisnika za poslovnu suradnju
* Kreiranje vježbi
* Kreiranje treninga
* Kreiranje rasporeda
* Pregled vježbi
* Pregled treninga
* Pregled rasporeda za pojedinog korisnika
* Mijenjanje profilne slike

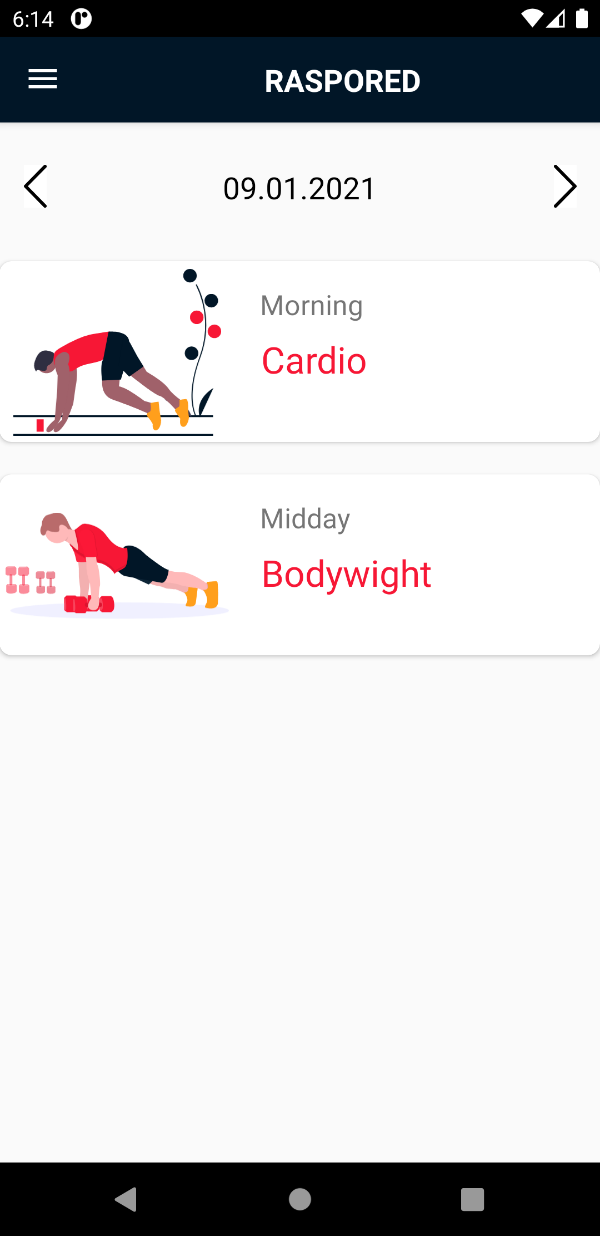
Trener putem aplikacije može zaraditi dodatan prihod. Količina prihoda ovisi o njihovom znanju, motiviranosti i konzistentnosti. Cilj je da aplikacija za trenere bude što jednostavnija za koristiti, bez nepotrebnih koraka ili duplih koraka. Stoga se sve kreirane vježbe i treninzi spremaju kako bi se mogli ponovno upotrijebiti.

#### Korisnik usluga trenera

Korisnici usluga koriste trenerova stručna znanja kako bi mogli kvalitetno odraditi svoje treninge. Za te usluge u dogovoru s trenerom s kojim imaju sklopljenu poslovnu suradnju ga plaćaju. Korisnici usluga imaju pristup sljedećim funkcionalnostima:

* Pregled vlastitog profila
* Pregled profila svog trenera
* Pregled rasporeda
* Pregled pojedinog treninga
* Pregled pojedine vježbe
* Gledanje videa vježbe
* Plaćanje

Cilj je da korisnik čim otvori aplikaciju dođe do željenih informacija, stoga prvi ekran koji se pokazuje je pregled rasporeda za taj dan.



Slika . Ekran korisnika usluga za pregled rasporeda

# Analiza uspješnosti aplikacija nakon produkcije

Nakon produkcije, aplikacija je dana na testiranje osobama koje su sudjelovale u anketi prije početka izrade sustava. Povratna informacija je pozitivna. Trenerima se dojmilo jednostavno slaganje rasporeda, treninga i vježbi, te jednostavnost postavljanja video zapisa uz vježbu. Reagirali su dobro na ideju da u skladu s korisnikom sami biraju iznos i vrijeme za plaćanje.

Korisnicima se izuzetno svidjelo jednostavan pregled rasporeda, treninga i vježbi. Posebno su pozitivno istaknuli funkcionalnost video zapisa uz svaku vježbu kao najveći doprinos ove aplikacije, jer su to personalizirani video zapisi snimljeni od strane trenera kako se koja vježba pravilno izvodi.

# Zaključak

Sustav *RemoteTrainer* izrađen je u svom osnovnom obliku sa svim glavnim funkcionalnostima, od kojih su mnogi bili sugerirani od strane anketiranih ispitanika. Pokazao se kao sustav s visokim potencijalom u današnje vrijeme kada postoji potreba za inovativnim rješenjima u potpuno novim situacijama.

Mogućnost treniranja na daljinu postala je sve potrebnija, ali bit će tražena i nakon povratka dnevnog života na stanje prije pandemije. Mogućnost više klijenata, samim time i više prihoda će privući sve više trenera na korištenje ovakvog načina treniranja. Istovremeno, manji troškovi za stručne naputke i treniranje iz udobnosti doma pokazali su se poželjnim od strane korisnika.

Analiza je potvrdila potencijal. Testna grupa od nekolicine trenera i korisnika pokazala je interes i zadovoljstvo nakon korištenja aplikacije u testnim uvjetima. Dokazano je da je ovakav način provođenja treninga jednako uspješan kao i uz izravan kontakt, dok god postoji volja i želja za treningom.

Baza podataka i web servis djeluju zasebno što omogućuje većem broju klijenata jednostavno povezivanje u isto vrijeme. Azure servis čini bazu podataka i web servis uvijek dostupnima i lako održivima, zato što Azure rješava pitanje dodatne IT infrastruktura.

Također nezavisnost baze podataka i web servisa omogućuje laku implementaciju novog klijenta za drugu vrstu uređaja u postojeći sustav.

Određene funkcionalnosti pokazale su se kao važniji dijelovi klijentskog sloja. Među njih spadaju video uz vježbu, pregled rasporeda, sastavljanje rasporeda treninga i vježbe. Za te funkcionalnosti je izuzetno bitan jednostavan dizajn korisničkog sučelja, kao i lako pregledavanje navedenih funkcionalnosti.

# Popis kratica

ACID *atomicity, consistency, isolation, durability* svojstva transakcije baze podataka

API *Application Programming Interface* aplikacijsko programsko sučelje

ART *Android runtime* android upravljač pokretanja

CRUD *create,read,update,delete* operacije nad tablicom baze podataka

CS *connection strin*g tekst za pristup bazi podataka

EF  *Entity framework* tehnologija za mapiranje podataka

HTTP  *Hyper-Text Transfer Protocol* protokol za razmjenu podataka

IT *information technology* informacijske tehnologije

JSON *JavaScript Object Notation* standard za razmjenu podataka

OLTP *Online transaction processing* mrežna obrada transakcija

ORM *object-relational mapping* objektno relacijsko mapiranje

RPC  *remote procedure call* pozivanje metoda na daljinu

SQL *Structured Query Language* strukturirani jezik upita

T-SQL *Transaction-Structured Query Language* jezik upita strukturiran transakcijama

URL *Uniform Resource Locator* web adresa

# Popis slika

[Slika 2.1 Prvo pitanje za trenere iz ankete 5](#_Toc62541395)

[Slika 2.2 Drugo pitanje za trenere iz ankete 5](#_Toc62541396)

[Slika 2.3 Treće pitanje za trenere iz ankete 6](#_Toc62541397)

[Slika 2.4 Četvrto pitanje za trenere iz ankete 6](#_Toc62541398)

[Slika 2.5 Peto pitanje za trenere iz ankete 7](#_Toc62541399)

[Slika 3.1 Životni ciklus aktivnosti [3] 12](#_Toc62541400)

[Slika 4.1 Arhitektura sustava *RemoteTrainer* 15](#_Toc62541401)

[Slika 4.2 Arhitektura baze podataka [13] 18](#_Toc62541402)

[Slika 4.3 Komunikacija web servisa [15] 21](#_Toc62541403)

[Slika 5.1 Relacijski model baze podataka 26](#_Toc62541404)

[Slika 5.2 Prikaz kreiranja prva tri entiteta. 27](#_Toc62541405)

[Slika 5.3 Prikaz izbornika za odabir način kreiranja modela 28](#_Toc62541406)

[Slika 5.4 Prikaz izbornika za odabir tablica tijekom kreiranja modela 29](#_Toc62541407)

[Slika 5.5 Izgled zaglavlja izbornika 34](#_Toc62541408)

[Slika 5.6 Izgled popisa opcija izbornika 35](#_Toc62541409)

[Slika 5.7 Izgled izbornika za trenera 35](#_Toc62541410)

[Slika 5.8 Izgled ekrana pronađi trenera 36](#_Toc62541411)

[Slika 5.9 Ekran trenera s korisnicima koji su poslali zahtjev za poslovnu suradnju treneru 39](#_Toc62541412)

[Slika 5.10 Ekran trenera za kreiranje treninga 40](#_Toc62541413)

[Slika 5.11 Ekran trenera za pregled vlastitih izrađenih treninga 41](#_Toc62541414)

[Slika 5.12 Ekran trenera za kreiranje rasporeda 42](#_Toc62541415)

[Slika 5.13 Ekran korisnika usluga za pregled rasporeda 44](#_Toc62541416)

# Popis tablica

[Tablica 4.1 Usporedba tankog i debelog klijenta [16] 22](#_Toc61859329)

# Popis kôdova

[Kôd 5.1 Prikaz kôda nakon kreiranja modela 30](#_Toc61859330)

[Kôd 5.2 API pristupna točka 32](#_Toc61859331)

[Kôd 5.3 Registriranje korisnika i dodavanje korisnika u bazu podataka 33](#_Toc61859332)

[Kôd 5.4 *menu\_items.xml* za korisnika 34](#_Toc61859333)

[Kôd 5.5 Prikaz svih trenera s *RecyclerView*  komponentom 37](#_Toc61859334)

[Kôd 5.6 Zahtjev poslovnoj sloja za dohvaćanje trenera. 38](#_Toc61859335)

[Kôd 5.7 Web servis metoda za dohvaćanje svih trenera 38](#_Toc61859336)

# Literatura

Svaki autor piše popis literature na kraju rada. Popis literature se piše stilom literatura.

1. Techopedia, What is Android Platform?

<https://www.techopedia.com/definition/4219/android-platform>, siječanj 2021.

1. Android guide  
   [https://developer.android.com/guide](https://developer.android.com/guide/components/services), siječanj 2021.
2. Android activity  
   <https://www.tutorialspoint.com/android/android_acitivities.htm> siječanj 2021.
3. LERMAN, J.; Programming Entity Framework: *Building Data Centric Apps with the ADO.NET Entity Framework*; O'Reilly Media; 2010
4. TechTarget, What is a Microsoft SQL Server?  
   <https://searchsqlserver.techtarget.com/definition/SQL-Server>, siječanj 2021.
5. Phillips, B., Hardy, B.; *Android Programming: The Big Nerd Ranch Guide;* Pearson Technology Group; 2017
6. Group; 2017 Rialdi, R., Sharma, A.,Bancila M*.; Learn C# Programming: a guide to building a solid foundation in C# language for writing efficient programs*. Packt Publishing; 2020
7. <https://www.c-sharpcorner.com/> siječanj 2021.
8. <https://en.wikipedia.org/wiki/Microsoft_SQL_Server> siječanj 2021.
9. [https://en.wikipedia.org/wiki/Interoperability#cite\_note-1](https://en.wikipedia.org/wiki/Interoperability%23cite_note-1) siječanj 2021.
10. <https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/> siječanj 2021.
11. <https://www.oracle.com/database/what-is-database/> siječanj 2021.
12. <https://www.srce.unizg.hr/files/srce/docs/edu/osnovni-tecajevi/d310_polaznik.pdf> siječanj 2021.
13. <https://en.wikipedia.org/wiki/ACID> siječanj 2021.
14. <https://www.tutorialsteacher.com/webapi/what-is-web-api> siječanj 2021.
15. <https://www.webopedia.com/insights/thin-vs-thick-client/> siječanj 2021.

Pristupnik: Filip Besednik, 0321007379

Mentor: struč.spec.ing.comp Daniel Bele

Datum: 02. 02. 2006.

**IMPLEMENTACIJA TJEŠENJA ZA VOĐENJE I IZVOĐENJE TRENINGA NA UDALJENOSTI**

